Ministère de L'éducation Nationale

UNIVERSITE MONTPELLIER II
UFR

MASTER 1

RAPPORT DE STAGE

Effectué au

CIRAD Persyst UPR 13

du 13 juin au 30 Aout 2007

Projet TSIGANE

Systeme d'information géographique
Accessible par l'internet

Rapport Technique

Par

Pierre LEMAN

Directeur de Stage :
Jean Baptiste LAURENT

Remerciements

Je tiens tout d'abord à remercier M.Jean-baptiste LAURENT mon tuteur de stage qui grâce à ses conseils, son aide et son soutien m'a permis de mener à bien mon stage. De même que M Pierre TODOROFF qui sont aussi des acteurs importants de ce projet.

Je remercie également M Eric BOURREAU pour sa disponibilité. Je remercie aussi toute l'équipe de MABIS de l'unité de recherche biostatistique, Philippe LETOURMY, Michel GINER, Eric GOZE, pour son accueil chaleureux et son soutien qui m'a permis de m'intégrer plus facilement.

Je remercie enfin les professeurs qui m'ont permis d'acquérir la méthode et les connaissances nécessaires à la réalisation de ce stage.

Glossaire

ODBC:

Open Database Connectivity. Couche Logicielle devant permettre à une application Windows d'accéder de façon transparente à une base de données SQL.

FTP : File Transfer Protocol. Protocole de transfert de fichier.

PHP : Langage de programmation open source Server Side

MVC : Architecture logicielle définie par la séparation de 3 entités indépendantes d'un logiciel : Modele-Vue-Controleur , ou le modèle définit les traitements , la Vue définit l'IHM et le Contrôleur lie les deux.

IHM : Interface homme-machine

API : Application and programming interface

GPL : general Public Licence

GIS : Systeme d'information geographique

IGN: Institut Géographique Nationnal

OGC : Open Gis Consortium

SIG: Systemes d'information Geographique

SQL: Structured Query Language

SVG: Scalable Vector Graphic

XML : Langage de balise extensible

1. Normes de programmation

Convention d'affectation de noms

Les conventions d'affectation de noms aident les programmeurs à :

- Normaliser la structure, le style de codage et la logique d'une application.
- Créer un code source précis, lisible et non ambigu.
- Etre cohérent entre les langages (Visual basic, C++, PHP…).
- Etre efficace du point de vue de la longueur des chaînes, offrant ainsi plus de possibilités pour des noms d'objets plus longs et plus complets.

Nommage des variables et des contrôles
Rien n'est complètement rigide, le nommage doit simplement être clair et non ambigu.

1. Comment nomme-t-on les variables ?

Les noms des variables sont structurés de la manière suivante :

[p][t]NomVariable (Exemple : gsCodeClient) **:**

- la portée [p]
- le type de la valeur de retour [t]
- un nom en toute lettre clair et concis

a. Les portées :

Locale	<Aucun>
Globale / Statique	g
Paramètre	p
Membre de la classe	m_
Constante	<Voir la section « 2. » plus loin>

Règles communes :

- Une variable ne peut possèder qu'une seule portée à la fois.
- Les portées sont toujours représentées en minuscule avant le type et le nom.
- La portée est optionnelle (si elle n'est pas précisée, la variable est locale).

b. Les types :

Array	a
Button	btn
Boolean	b
EditBox	edit
CheckBox	chk
Combo	cbo
Image	img
Integer	n

Label	lbl
Long	l
Object	o
String	s
Variant	v

Règles communes :

- Une variable ne peut possèder qu'un seul type à la fois exception faite pour le tableau de string qui utilisera le préfixe « as » sinon utilisez le type « v » (Variant).
- Les types sont toujours représentés en minuscule après la portée et avant le nom.
- Le type est obligatoire.

c. Les noms :

Le nom de la variable renseigne sur son contenu.

Règles communes :

- Un nom de variable utilise une majuscule pour chaque début de mot.
- Une variable ne doit pas stocker deux contenus différents (Créez une autre variable, ne réutilisez pas celle existante pour économiser.)
- Les noms sont écrits en toutes lettres.
- Le nom est obligatoire.
- N'utilisez pas de séparateur comme « _ »

Exemples de variables :

$gsCodeClient	Variable globale de type String stockant un code client
$m_lCodeClient	Membre de la classe de type Long stockant un code client
$editCodeClient	Objet local de type EditBox stockant un code client

2. Les constantes

Les noms des constantes :

- sont entièrement en majuscule.
- commencent par « CONST ».
- précisent en premier lieu le composant concerné.
- sont découpés grâce au caractère « _ »

Exemple de constantes :

CONST_USER_ADMINGROUP	Constante du composant USER décrivant le groupe d'administration.
CONST_PAGE_DEFAULTCSS	Constante de page décrivant le style CSS par défaut

3. Les fonctions

Les noms de fonction commencent par une majuscule.
Chaque mot constituant le nom commencent par une lettre majuscule.
Le caractère « _ » en début de nom précise si la fonction est « private ».
Exemple de fonction :

GetHTML()	Fonction retournant une représentation HTML
_SetFocus(psFormName, psFieldName)	Fonction private donnant le focus à un champ de formulaire

La présentation du code Source

Votre code Source sera lu et relu, par vous même tout d'abord mais également par vos collaborateurs. Le jour même de sa réalisation et parfois plusieurs mois après.

Aussi il est important de soigner votre source.

1. **Aérez votre source :**
   - Sautez des lignes
   - Ajoutez des tabulations pour représenter les imbrications
   - Ne saisissez pas plus d'une opération par ligne
   - Découpez les fonctions en sous fonction si necessaire (Une fonction doit faire idéalement 10 lignes de code maximum hors commentaire).

2. L'appel de fonctions

Les fonctions doivent être appelées sans espace entre le nom de la fonction, la parenthèse ouvrante, et le premier paramètre.
Il doit y avoir un espace de chaque côté du signe égal utilisé pour affecter la valeur de retour de la fonction à une variable.
Dans le cas d'un bloc d'instructions similaires, des espaces supplémentaires peuvent être ajoutés pour améliorer la lisibilité.
Exemple de code :

```php
<?php
$var   = foo($bar, $baz, $quux);
$short = foo($bar);

?>
```

3. Déclaration et Structure de Contrôle

La déclaration des fonctions respecte l'indentation classique des accolades :

```php
<?php
function fooFunction($arg1, $arg2 = '')
{
    if (condition) {
        statement;
    }
    return $val;
}
?>
```

Les arguments possédant des valeurs par défaut vont à la fin de la liste des arguments. Il faut toujours chercher à retourner une valeur ayant un sens lorsque cela est possible.

Les structures de contrôles incluent les 'if', 'for', 'while', 'switch', etc. Vous trouverez ici un exemple de structure 'if' qui est la plus compliquée :

```
<?php
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
}
?>
```

Les instructions de contrôle doivent avoir un espace entre le mot clé de l'instruction et la parenthèse ouvrante, afin de les distinguer des appels de fonctions.

Il est vivement recommandé de toujours utiliser des accolades, même dans les situations où elles sont techniquement optionnelles. Leur présence augmente la lisibilité du code et réduit le risque d'erreur logique lors de l'ajout de nouvelles lignes de code.

Pour l'instruction 'switch' :

```
<?php
switch (condition) {
case 1:
    action1;
    break;

case 2:
    action2;
    break;

default:
    defaultaction;
    break;
}
?>
```

4. Portabilité

Utilisez *toujours* <?php ?> et non pas <? ?>

5. **Les standards Docblock pour les commentaires**

6. **Rendez votre source accessible :**
   - Commentez chaque fonction à l'aide de phpDoc (Objectif, Paramètre, valeur de retour) http://pear.php.net/manual/fr/standards.header.php
   - Commentez les lignes obscures
   - Utilisez les règles de nommages définies
   - Utilisez des noms clairs et concis

2. Analyse – Conception

Conception du requêteur

Le requêteur est une des parties du portail d'accès Web du projet tsigane. Ce dernier est dédié à la sélection d'un ensemble de parcelles correspondantes à un ensemble de contraintes prédéfinies que l'utilisateur peut utiliser ou non pour effectuer une requête sur les données de la base (d'où le terme de requêteur).

Le requêteur est divisé en deux parties conceptuellement distinctes : L'une pour la sélection et la saisie des critères de recherche et l'autre pour l'affichage des résultats.

3. Réalisation

Aspects Web

Réalisation du requêteur (compatible PHP4/PHP5)

La conception générale du requêteur a suivit les principes directeurs suivant :

- **1 Formulaire = 1 instance d'objet formulaire**

La conception du formulaire suit les principes de bases de la programmation orientée objet qui veut que l'on regroupe à la fois les données et les traitements sur les données dans un même objet. C'est ici le cas étant donné que la classe Formulaire encapsule à la fois les données de base du formulaire (La page php cible du formulaire par exemple) et le traitement sur ces données (par exemple, les différents générateurs de code qui permettent de construire dynamiquement la page).

On aura donc pour chaque page de formulaire exactement un objet formulaire. A chaque action nécessitant un rechargement de la page, ce dernier est sérialisé et stocké dans une variable $_POST puis désérialisé au rechargement de la page. L'objet est par conséquent transmis de la page à elle-même et la persistance de l'objet est maintenue. L'utilisation de ce mécanisme est des plus simples :

```php
<?php require_once("formulaire.class.php");

	$oFormulaire = "";
	//Si le formulaire n'a pas deja été instancié
	if (!isset($_POST['Formulaire'])) {
		$oFormulaire = new Formulaire("requete.php");
	}
	else {
		//Restauration de l'objet sérialisé
		$oFormulaire = unserialize(urldecode($_POST['Formulaire']));
	}

?>
```

- **3 méthodes d'affichage pour trois éléments à positionner différemment**

Une fois instancié, le formulaire s'utilise par l'intermédiaire de trois méthodes publiques (non préfixées par _ par convention) permettant d'afficher les trois éléments que sont la zone de saisie et de validation des critères de recherche, la zone de visualisation des résultats, ainsi que la zone permettant la navigation dans les pages de résultats de recherche (seulement 100 résultats de recherche sont présentés sur chaque page).

Les raisons de ce découpage ne sont pas techniques mais pratiques : afin de conserver la charte graphique du site telle que définie par le graphiste qui en est à l'origine, il fallait insérer le code HTML généré pour le formulaire aux bons endroits afin de conserver l'aspect général, ce qui fut fait.

Signature des trois méthodes :

`afficheFormulaireCriteres()` : zone de saisie et validation des critères de recherche
`afficherResultatRecherche()` : zone de visualisation des résultats
`afficheFlechesNavigation()` : zone de navigation dans les pages de résultats de recherche

- **1 unique formulaire <form> pour tout le formulaire**

Le nombre de formulaires <form> générés par la classe Formulaire a été volontairement limitée à un afin d'assurer les fonctionnalités suivantes :

- Pré-remplissage des champs déjà saisis par l'utilisateur lors de la précédente recherche ou d'une action quelconque nécessitant le rechargement du formulaire. En cas de formulaires multiples, il devient impossible de récupérer la valeur de tous les champs saisis par l'utilisateur pour la simple bonne raison qu'ils ne sont pas passés en paramètre car pas dans le formulaire courant.
- Tous les formulaires auraient eu la même cible car la classe Formulaire est conçue de manière à être utilisée en tandem avec la page php qui l'instancie (cette dernière constituant la cible du formulaire instancié).

Paramétrage du formulaire

Depuis son origine la classe Formulaire a été conçue de manière à être la plus générique et donc réutilisable possible. Bien que d'avantage spécialisée que la version intermédiaire, la version finale est encore facilement réutilisable.

En effet, la plupart des données spécialisant le formulaire telles que les tables de la base cible du requêteur ou encore la page php cible sont contenues dans des variables membres de la classe Formulaire qui sont transmises et donc conservées à chaque rechargement de la page par le mécanisme de sérialisation.

Dans la version finale du requêteur, ces données sont renseignées une bonne fois pour toutes dans le constructeur la classe Formulaire.

Dans la suite nous présentons les variables à paramétrer obligatoirement. Plus d'informations sur ces variables peuvent être trouvées directement dans le code source.

`$m_sTableCible` : Table cible du formulaire dans la base de données.
`$m_sClePrimaireTableCible` : Clé primaire de la table cible du formulaire.
`$m_sNomArchiveGE` : Indique le nom que portera l'archive Google Earth générée par le générateur de code KML.
`$m_sNomDossierGE` : Indique le nom du dossier Google Earth qui sera généré pour contenir les informations.
`$m_sInformationsGE` : Indique les informations liées aux données Google Earth
$m_aAlias : Indique le contenu de la table des alias, c'est à dire des correspondances entre nom dans la table et nom affiché
Utilisé pour l'affichage à destination de l'utilisateur final.
`$m_aListeCriteresFormulaire` : Indique les critères qui figureront dans l'interface des critères de recherche
`$m_aDonneesClasse` : Indique la liste des champs de la table qui devront être affichés sous forme de classe de valeurs
`$m_aDonneesListe` : Indique la liste des champs de la table qui devront être affichés sous forme de liste de choix de valeurs

`$m_aBlackList` : Indique la liste des champs de la table (critères) qui devront être blacklistés, c'est-à-dire qu'ils ne devront jamais apparaître dans les résultats de recherche. Ces derniers sont volontairement cachés à l'utilisateur car considérés comme non désirables (exemple : les champs géométriques propre au SGBD spatial utilisé).
$m_aChampsAAfficherOffice : Permet de connaître la liste des champs à afficher d'office dans le tableau présentant les résultats de la requête.
Les autres champs seront affichés s'il reste suffisamment de place, sinon ils resteront cachés sauf si l'utilisateur clique sur l'icône de demande d'informations (la croix).
`$m_aChampsAdditionnels` : Tableau associatif indiquant des critères de recherche ne reposant pas sur des critères de la base. C'est par exemple le cas pour certaines informations de type géographique qui seront générées en temps réel par le module postGIS de postgreSQL.

Aspects BD

La classe Acces_BD

Cette classe assure la gestion de tout ce qui concerne la connexion et l'accès à la base de données postgreSQL. Elle assure en particulier les points suivant :

- Prise en charge des erreurs : les erreurs pouvant survenir sont capturées et traitées sous la forme d'un message d'erreur de retour à destination de l'utilisateur.

Paramétrage de la classe Acces_BD

A l'instar de la classe Formulaire, la classe Acces_BD est paramétrable en ce qui concerne certaines informations telles que le nom (ou bien l'adresse IP) de l'hôte et le numéro de port pour la connexion.

Nous présentons ci-dessous la liste des paramètres devant être passés au constructeur afin de se connecter à la base de données de son choix. Des valeurs par défaut sont déjà prédéfinies ce qui fait que le passage des arguments au constructeur est facultatif.

Pour des raisons de simplicité évidentes, nous encourageons le développeur à définir des valeurs préfinies pour chacun des paramètres puis de faire des appels simples sans les arguments.
$psHost : Le serveur hébergeant de la base de données.
$psPort : Le port d'écoute du serveur.
$psDbName : Le nom de la base pour la connexion.
$psUser : Le login utilisateur sur la base de données.
$psPassword : Le mot de passe utilisateur pour la base de données.

Les tables additionnelles à rajouter à la base Géoclip

La table f_parc_bck

Cette table, copie de la table f_parc déjà présente dans Géoclip, est utilisée afin de conserver un backup de l'ensemble des parcelles de Géoclip. Cette nécessité provient du fait que l'on écrase à chaque requête de l'utilisateur le contenu de cette table afin de voir directement le résultat de la requête de sélection dans Géoclip, il faut donc conserver la source de données intacte quelque part.

On sélectionne ensuite les tuples de cette table backup qui correspondent aux critères par jointure avec la table tmp_parcelles, cette dernière servant de stockage pour les clés primaires des tuples de la table parcelles correspondantes aux critères de recherche. Il n'y donc plus ensuite qu'a écraser le contenu de la table f_parc par le résultat de cette jointure.

Code SQL :
CREATE TABLE f_parc_bck

```
(
  objectid integer,
  shape geometry,
  _index_minx integer,
  _index_miny integer,
  _index_maxx integer,
  _index_maxy integer,
  codgeo character varying(10) NOT NULL,
  num_pacage integer,
  surface double precision,
  libgeo character varying(50),
  CONSTRAINT pk_f_parc_old PRIMARY KEY (codgeo)
)
```

Les tables parcelles et tmp_parcelles

Ces deux tables, spécifiques au requêteur, servent à stocker des données complètes relatives aux parcelles. Leur justification tient dans le fait que bien que Géoclip contiennent dans ses tables de base la plupart des informations nécessaires, certaines manquent (par exemple, le nom du propriétaire de la parcelle), et d'autre part car ces données n'était pas centralisées et obligeait donc à des jointures coûteuses et complexe à mettre en œuvre.

Code SQL de création de la table parcelles :

```
CREATE TABLE parcelles
(
  num_parcelle integer NOT NULL,
  ref_cadastrale character varying,
  nom character varying,
  localisation character varying,
  num_pacage integer,
  pente real,
  altitude integer,
  statut_foncier smallint,
  ancien_num_surf_geographique_elementaire integer,
  source_creation character varying(5),
  description text,
  date_saisie_source date,
  origine_ebauche text,
  commune integer,
  latitude double precision,
  longitude double precision,
  geom geometry,
  surface_declare numeric(10,2),
  nom_exploitant character varying,
  culture integer,
  rendement integer,
  CONSTRAINT pk_parcelles PRIMARY KEY (num_parcelle),
  CONSTRAINT "enforce_dims_GEOM" CHECK (ndims(geom) = 2),
  CONSTRAINT enforce_srid_geom CHECK (srid(geom) = 32620)
)
```

Note : remarquer ici l'importance des deux dernières contraintes de type géométrique et prises en charges par PostGIS. La première enforce_dims_GEOM vérifie que le champ géométrique geom de toute parcelle dont le champ est renseigné est de dimension deux et la deuxième que le SRID du champ géométrique soit exactement égal à 32620.

Ces deux contraintes sont capitales car permettent la normalisation des données géométriques de la table parcelles en rendant impossible l'insertion de données non conformes. Les parcelles seront donc toujours représentable dans le plan (en dimension

deux) et indexées au bon SRID (celui correspondant au système de coordonnées géoréférencées utilisé pour les mesures).

De nombreux tests peuvent cependant être ajoutés telles qu'un test de validité des nouvelles parcelles (fonction isvalid() de PostGIS) ou encore la vérification que la nouvelle parcelles se situe bien sur le territoire de la Guadeloupe (fonction contains() de PostGIS).

Code SQL de création de la table tmp_parcelles :

```
CREATE TABLE tmp_parcelles
(
  num_parcelle integer NOT NULL,
  CONSTRAINT pk_tmp_parcelles PRIMARY KEY (num_parcelle)
)
WITHOUT OIDS;
ALTER TABLE tmp_parcelles OWNER TO postgres;
```

Aspects SIG

Architecture de la base de données spatiale : PostgreSQL et les modules d'extension spatiale

PostgreSQL utilisé seul fourni les services d'un SGBD classique : des modules d'extensions permettent cependant de le transformer en véritable SGBDS ou SGBD spatial :

- PostGIS rajoute à PostgreSQL le support des objets spatiaux, qui peuvent des objets géométriques de toute forme ou dimension.
- Proj : librairie de projection cartographique permettant d'étendre PostGIS en rajoutant la fonctionnalité de reprojection permettant de passer d'un système de coordonnées géoréférencées à un autre. Très utilisé dans le requêteur afin de convertir les données géométriques de tsigane (dans le système WGS84 / UTM zone 20'' N de SRID 32620) vers le système de coordonnées de Google Earth WGS84 de SRID 4326.
- Geos : La bibliothèque GEOS est utilisée pour permettre l'utilisation des tests géométriques (Touches(), Contains(), Intersects()) et les opérations (Buffer(), GeomUnion(), Difference()) avec PostGIS. Cette bibliothèque n'est pas pour l'instant vraiment utilisée mais présente un fort potentiel.

Représentation des informations géographiques de tsigane

Les données géographiques à représenter étaient de type parcelle agricole et stations météorologique. Dans les deux cas, le mode de représentation choisi est assez intuitif :

- Pour la représentation des parcelles, nous avons utilisé le type POLYGON de PostGIS qui permet de définir des polygones de n'importe quelle dimension. Afin d'éviter la saisie de parcelles et donc de polygones de n'importe quelle dimension, nous y avons adjoint une contrainte sur la dimension des polygones : cette dernière doit pour être acceptée être égale exactement à deux. Nous considérons que les parcelles doivent pouvoir être représentées dans le plan.
  Nous avons rajouté une deuxième contrainte à celle-ci : étant donné que le système de coordonnées géoréférencées utilisé pour la saisie des données géographique dans tsigane est connu (WGS84 / UTM zone 20'' N de SRID 32620), nous obligeons à ce que les nouveaux objets géographiques soient représentés dans ce système là. Cette partie est expliquée plus en détails dans la partie sur les aspects base de données de tsigane.
- En ce qui concerne la représentation des stations météorologiques, nous avons utilisé le type POINT de PostGIS qui permet de représenter facilement un objet ponctuel.

Comme précédemment, nous définissons des contraintes sur la dimension (nous considérons ici aussi qu'une station météo doit pouvoir être représenté dans le plan) et sur le système de coordonnées utilisées (le même que précédemment).

Aspects génération de code/exportation des données géographiques

Nous présentons dans cette partie la possible exportation et donc réutilisation des données de tsigane dans des applications tierces non spécifiques au projet tsigane.

Générateur KML pour exportation vers Google Earth/Maps

Le langage KML

KML (Keyhole Markup Language) est un langage basé sur le formalisme XML et destiné à la gestion de l'affichage de données géospatiales dans les logiciels Google Earth, Google maps, Google Mobile et World Wind.
Les fichiers KML peuvent également se présenter avec l'extension .kmz qui est la version zippée du fichier KML.

Les spécifications du langage peuvent être trouvées à l'adresse :
http://code.google.com/apis/kml/documentation/index.html
A noter qu'un validateur de document KML à fait son apparition. Il est disponible à l'adresse :
http://feedvalidator.org/

La classe GenerateurKML

Cette classe prend en charge l'intégralité du processus de transformation depuis les données spatiales contenues dans la base jusqu'à l'obtention du fichier KML.

Le processus de transformation est découpé en sous-étapes que l'on a nommé respectivement génération de l'en-tête, génération du corps, génération du pied de page et pour finir l'enregistrement à proprement parler du document sous la forme d'un document compressé KML d'extension KMZ.

L'en-tête d'un document KML contient la définition des styles (par exemple, en ce qui concerne les parcelles, on peut changer les couleurs de représentation, la taille des bordures, etc …). On défini notamment des informations telles que le point de centrage et le niveau de zoom de la vue par défaut lors du chargement du fichier (dans le cas d'un parcellaire Guadeloupéen, par exemple, on aura tout intérêt à centrer la vue sur la Guadeloupe). Des informations supplémentaires telles que le nom de l'archive qui contiendra le parcellaire ou encore le propriétaire des données (ici, le CIRAD) semblent aussi indiquées.

Le corps du document KML est lui l'élément central d'un document KML étant donné qu'il contient l'intégralité des données géographiques ainsi que d'autres liées aux parcelles. Ces données géographiques en langage KML sont la résultante d'un processus de transformation en deux étapes :

- Dans un premier temps, on convertit les données de la base du type Geometry (le type des données géométriques de PostGIS) vers le format GML à l'aide de la fonction asgml() de PostGIS. Cette fonction prend en paramètre une donnée binaire de type Geometry et renvoi son équivalent textuel dans le langage de description GML.
- Dans un second temps, on fait appel à un convertisseur du format GML vers le format KML. Ces deux langages sont tous deux textuels et sont équivalent au nom des balises près ce qui rend la conversion facile. Bien qu'une fonction d'export direct vers le format KML ai vue le jour dans les dernières versions de PostGIS, il a été décidé de ne pas l'utiliser afin d'assurer un maximum de rétrocompatibilité avec les anciennes versions. Le générateur KML actuel utilise la classe Gml2Kml pour réaliser cette conversion.
  Une optimisation simple sera un jour d'utiliser la fonction askml() pour réaliser cette transformation, beaucoup plus rapide car réalisée à l'intérieur même de la base de données.

Une fois toutes les données mises sous forme de document KML, il ne suffit plus que de refermer les balises, ce à quoi sert le pied du document KML.

La phase de génération terminée, et cette dernière s'exécutant totalement en mémoire vive, il faut ensuite le sauvegarder sous la forme d'un fichier que l'utilisateur final pourra télécharger (le but étant de lui permettre de l'ouvrir dans Google Earth). Google Earth accepte deux formats pour l'importation de données géographiques : soit directement KML, soit la forme compressée au format ZIP de ce format, KMZ.

Différents tests menés ont montré que le temps de compaction vers ce format était négligeable à la vue du gain d'espace disque réalisé (en général d'un facteur 10).

Afin de réaliser cette compaction, nous utilisons une librairie PHP issue de phpMyAdmin, un projet visant à donner une interface graphique conviviale en accès Web à la base de données MySQL. Les sources de cette librairie sont situées dans le répertoire phpMyAdmin\libraries\zip.lib.php de phpMyAdmin et placées sous la licence libre GPL.

Le fichier KMZ obtenu au final n'est en fait d'un fichier KML zippé dans un fichier KMZ de même nom portant bien évidemment l'extension KMZ.

Ce fichier est ensuite écrit physiquement sur le disque dur du serveur afin d'être proposé au téléchargement à l'utilisateur, ce qui se manifeste par l'apparition d'un icône de téléchargement dans son interface Web.

Pour l'instant, la concurrence des requêtes n'est pas prise en compte ce qui fait que chaque demande de génération entraîne la suppression du fichier précédemment créé. Le système résultant est donc mono-utilisateur.

Générateur CSV pour exportation vers Office/OpenOffice/etc ...

Le langage CSV

CSV est un format informatique ouvert représentant des données tabulaires. CSV est l'abréviation de l'anglais « comma-separated values », valeurs séparées par des virgules. Ce format est utilisé en bureautique afin d'assurer un export facile des feuilles de calcul.

La méthode exporterVersCSV() de la classe Formulaire

Cette méthode de la classe Formulaire permet d'exporter les parcelles que l'utilisateur a préalablement cochées (dans l'interface du requêteur) vers un fichier CSV qui sera ensuite exploitable par exemple à l'aide d'une suite bureautique telle que la suite Office.

Le document CSV étant constitué de deux parties, un en-tête contenant le nom des champs retenus pour exportation, et d'un corps contenant les données à proprement parler, l'exportation se déroule en deux phases :

1. Pour chaque nom d'en-tête, on le concatène avec les précédents sur la même ligne en les séparant par ;
2. Pour chaque case cochée par l'utilisateur dans l'interface du requêteur, on créé une nouvelle ligne et on concaténe la valeur de chaque variable séparée par ;

A la fin de ces deux opérations le contenu du fichier CSV est écrit dans un fichier de nom prédéfini à l'instar du générateur de code KML.

4. Déploiement

Nous décrirons dans cette partie différentes manières de mettre en œuvre la solution développée (c'est-à-dire le portail du site Tsigane, auquel on adjoint l'outil Géoclip pour une visualisation directe des données spatiales dans Tsigane).

La première, plus rapide à mettre en œuvre, est effectuée sous Windows XP et sera utilisée principalement dans l'optique du développement et des tests. La deuxième, plus sécurisée mais aussi complexe à mettre en œuvre, conviendra davantage à la mise en production.

Mise en œuvre rapide sous Windows XP

Installation de PostgreSQL + PostGIS + Proj + Geos

1. Télécharger tout d'abord l'archive postgresql-X.zip sur le site :
   http://pginstaller.projects.postgresql.org/
   Ce lien contient aussi un tutoriel en anglais qui peut se révéler utile afin de comprendre certaine spécificités de l'installateur.
2. Décompacter l'archive et lancer l'exécutable msi.
3. Choisir la langue et accepter la licence d'utilisation.
4. Dans la fenêtre « Feature selection », veiller a bien conserver sélectionnée l'option « PostGIS Spatial Extension » afin d'installer l'extension spatiale de PostgreSQL. Les autres options sont laissées à appréciation mais nous recommandons au minimum les options « Répertoire des données », « psql » et « pgAdmin III ».
5. La fenêtre « Configuration du service » propose l'installation ou non du SGBD PostgreSQL comme un service du système. Si PostgreSQL doit être démarré en même temps que le système, cocher cette option et donner les informations nécessaires à la création d'un compte de service pour lancer le service PostgreSQL.
6. Dans la fenêtre « Initialisation du groupe de bases de données », laisser cochée la case « Initialiser le groupe de base de données ».
   Si l'installation courante de PostgreSQL est la première sur le système, garder le port 5432 par défaut.
   Si PostgreSQL ne doit être accessible que depuis l'ordinateur ou il est installé laisser décochée la case « Accepte les connexions sur toutes les adresses, pas seulement localhost ».
   Dans le menu déroulant du choix des locales choisir « French, France », et sélectionner le jeu de caractères « SQL_ASCII » du menu de choix du codage.
   Choisir enfin un identifiant et un mot de passe pour le compte du Superutilisateur(DBA).
7. Dans la fenêtre « Activation des langages de procédures », sélectionner au minimum le langage PL/pgsql. Ce dernier est en effet utilisé par certaines des fonctions de PostGIS et est donc indispensable à son fonctionnement.
8. Dans la fenêtre « Activation des modules de contribution », laisser les choix par défaut.
9. Dans la fenêtre « Activer PostGIS » cocher la case « Activer PostGIS pour le template1 ».
10. Terminer l'installation.

Installation du serveur Web Apache via EasyPHP

1. Réaliser l'installation par défaut d'EasyPHP.
2. Activer le support de PostgreSQL dans Apache :
   a. Dans le fichier …\EasyPHPX.X\conf_files\php.ini décommenter la ligne :
      ;extension=php_pgsql.dll
      En supprimant le ; comme ceci :
      extension=php_pgsql.dll
   b. Dans le fichier …\EasyPHPX.X\apache\php.ini décommenter la ligne :
      ;extension=php_pgsql.dll
      Comme précédemment.

   Note : Attention, il existe 3 fichiers php.ini différents dans EasyPHP.
3. Autoriser l'accès au serveur Apache de EasyPHP depuis une machine distante :
   Par défaut l'accès au serveur Apache est limité à l'adresse loopback (localhost ou 127.0.0.1).
   Pour pouvoir y accéder depuis n'importe quel hôte, il faut modifier le fichier de configuration d'Apache httpd.conf, et commenter la ligne :
   Listen 127.0.0.1:80
   En la changeant en :
   #Listen 127.0.0.1:80

   La doc complète ici : http://www.easyphp.org/intra-inter.php3

Note : Certains problèmes peuvent subsister avec le firewall inclus dans Windows XP SP2. Penser à débloquer le processus Apache voire à désactiver le firewall en cas de problèmes.

4. Installation de Géoclip dans Apache :
   Copier tout les fichiers de géoclip dans le repertoire www d'EasyPHP.
   Modifier le fichier \utils\conn.inc.php de manière à ce que le nom de la base, les identifiants, etc … correspondent à la base installée sur PostgreSQL.

Mise en production sous Debian Etch 4.0

Installation du serveur (Debian Etch 4.0)

Afin d'assurer une sécurité et une stabilité maximales, il est recommandé de n'installer que le système Debian de base (sans l'interface graphique, etc…) puis de rajouter uniquement les quelques paquets logiciels nécessaires (PostgreSQL, …).

Installation de Apache + PHP + Géoclip

1. Mise à jour des sources d'apt :
   Afin d'installer les paquets les plus récents, il est nécessaire de mettre à jour la liste des paquets de l'outil *apt*. Identifiez-vous en **root** dans un terminal à l'aide de la commande su. Tapez ensuite *apt-get update*. Une fois le téléchargement terminé, vous disposez de la liste des paquets les plus récents.
2. Installation du serveur Apache 2 :
   Il s'agit du serveur Web seul. Sans documentation, sans langage coté serveur, sans base de données.
   Dans une console tapper :
   apt-get install apache2
3. Installation de PHP avec prise en charge de PostgreSQL :
   Dans une console tapper :
   apt-get install libapache2-mod-php4 php4-pgsql php4-cli
   Dans le cadre d'une installation de PHP5 au lieu de PHP4 remplacer toutes les occurrences de 4 par 5.
4. Copier les fichiers sources PHP de Géoclip dans le dossier */var/www/*
   Modifier le fichier /var/www/utils/conn.inc.php de manière à ce que le nom de la base, les identifiants, etc … correspondent à la base installée sur PostgreSQL.

Configuration d'Apache et de PHP

Fichier httpd.conf (Apache)
Indiquer aux serveur Apache sur quels adresses IP et port il doit écouter : la directive Listen

La directive Listen enjoint Apache à écouter plus d'une adresse IP ou port; par défaut Apache répond aux requêtes reçues sur toutes les interfaces IP, mais seulement celles arrivant sur le port donné par la directive Port.

Listen peut être utilisée à la place de BindAdress et Port. Elle indique au serveur d'accepter des requêtes entrantes sur le port spécifié ou sur une combinaison adresse-port. Si le premier format est utilisé (avec seule mention d'un numéro de port), le serveur "écoutera" tous les ports spécifiés sur chacune des interfaces IP qu'il connaît, plutôt que sur le port donné par la directive Port. Si une adresse IP adresse IP est précisée en complément, le serveur restreindra son écoute à la combinaison adresse-port précisée.

Notez que vous avez toujours besoin de la directive Port qui permet à Apache de générer les URL de retour vers votre serveur.

Plusieurs directives Listen peuvent être utilisées pour spécifier un ensemble d'adresses et de ports à écouter. Le serveur répondra aux requêtes reçues sur n'importe laquelle des combinaisons adresse-port ainsi spécifiée.

Par exemple, pour autoriser le serveur à accepter des connexions sur les ports 80 et 8000, écrire :

Listen 80
Listen 8000

Pour autoriser un serveur à accepter des connexions sur deux "sockets" qualifiés, écrire :
Listen 192.170.2.1:80
Listen 192.170.2.5:8000

Fichier php.ini (PHP)

Certains scripts PHP étant particulièrement consommateurs en ressources et long en exécution (par exemple, le script de génération de code KML pour Google Earth), il peut fréquemment arriver que le script « plante » pour une raison de dépassement de délais d'exécution autorisé ou encore de dépassement de limite de ressource avant la fin de son exécution.

Ce genre de problème peut être résolu en éditant le fichier php.ini de PHP dans la section ressource limits et en augmentant la valeur des variables présentes.
Note : En général, mettre la variable max_execution_time à 0 suffit amplement à résoudre ce genre de problème (temps d'exécution infini possible).

;;;;;;;;;;;;;;;;;;
; Resource Limits ;
;;;;;;;;;;;;;;;;;;

max_execution_time = 0 ; Maximum execution time of each script, in seconds
max_input_time = 60 ; Maximum amount of time each script may spend parsing request data
memory_limit = 16M ; Maximum amount of memory a script may consume (8MB)

Installation de PostgreSQL + PostGIS + proj + geos

Dans une console taper :
apt-get install postgresql-8.1 postgresql-8.1-postgis proj libgeos-c1

Ceci va installer et préconfigurer tout les paquets nécessaires.

Configuration de PostgreSQL

PostgreSQL comporte trois fichiers de configuration dont deux principaux :
postgresql.conf : configuration générale de la base (port pour les connexions TCP/IP, …)

doc sympa : http://docs.postgresqlfr.org/annotated_postgresql_conf_81.html

pg_hba.conf : Droits d'accès à la base de données en fonction des critères local(Unix)/non local (TCP/IP).
La doc complète : http://docs.postgresqlfr.org/8.1/client-authentication.html

5. Annexes

Annexe A - Conventions de codage

Nommage des variables et des contrôles

Rien n'est complètement rigide, le nommage doit simplement être clair et non ambigu.

2. Comment nomme-t-on les variables ?

Les noms des variables sont structurés de la manière suivante :

[p][t]NomVariable (Exemple : gsCodeClient) **:**

- la portée [p]
- le type de la valeur de retour [t]
- un nom en toute lettre clair et concis

a. Les portées :

Locale	<Aucun>
Globale / Statique	g
Paramètre	p
Membre de la classe	m_
Constante	<Voir la section « 2. » plus loin>

Règles communes :

- Une variable ne peut possèder qu'une seule portée à la fois.
- Les portées sont toujours représentées en minuscule avant le type et le nom.
- La portée est optionnelle (si elle n'est pas précisée, la variable est locale).

b. Les types :

Array	a
Button	btn
Boolean	b
EditBox	edit
CheckBox	chk
Combo	cbo
Image	img
Integer	n
Label	lbl
Long	l
Object	o
String	s
Variant	v

Règles communes :

- Une variable ne peut possèder qu'un seul type à la fois exception faite pour le tableau de string qui utilisera le préfixe « as » sinon utilisez le type « v » (Variant).
- Les types sont toujours représentés en minuscule après la portée et avant le nom.
- Le type est obligatoire.

c. Les noms :

Le nom de la variable renseigne sur son contenu.

Règles communes :

- Un nom de variable utilise une majuscule pour chaque début de mot.
- Une variable ne doit pas stocker deux contenus différents (Créez une autre variable, ne réutilisez pas celle existante pour économiser.)
- Les noms sont écrits en toutes lettres.

- Le nom est obligatoire.
- N'utilisez pas de séparateur comme « _ »

Exemples de variables :

$gsCodeClient	Variable globale de type String stockant un code client
$m_lCodeClient	Membre de la classe de type Long stockant un code client
$editCodeClient	Objet local de type EditBox stockant un code client

4. **Les constantes**

Les noms des constantes :

- sont entièrement en majuscule.
- commencent par « CONST ».
- précisent en premier lieu le composant concerné.
- sont découpés grâce au caractère « _ »

Exemple de constantes :

CONST_USER_ADMINGROUP	Constante du composant USER décrivant le groupe d'administration.
CONST_PAGE_DEFAULTCSS	Constante de page décrivant le style CSS par défaut

5. **Les fonctions**

Les noms de fonction commencent par une majuscule.
Chaque mot constituant le nom commence par une lettre majuscule.
Le caractère « _ » en début de nom précise si la fonction est « private ».
Exemple de fonction :

GetHTML()	Fonction retournant une représentation HTML
_SetFocus(psFormName, psFieldName)	Fonction private donnant le focus à un champ de formulaire

Annexe B – Liste des variables POST du formulaire

Variables POST des boutons de soumission du formulaire

btnRechercher
Le bouton permettant de lancer une recherche avec les critères courants.

btnReset
Le bouton de remise à zéro du formulaire.

btnExportCSV
Le bouton permettant de lancer l'exportation des données liées aux cases cochées par l'utilisateur vers un fichier CSV.

btnExportGoogleEarth
Le bouton permettant de lancer l'exportation des données de toute la recherche courante vers un fichier KML lisible par Google Earth.

btnDebut
Le bouton permettant de se rendre à la première page de résultats de la recherche courante si cette dernière existe.

btnPrecedent
Le bouton permettant de passer à la page précédente de résultats si cette dernière existe.
btnSuivant
Le bouton permettant de passer à la page suivante de résultats si cette dernière existe.

btnFin
Le bouton permettant de se rendre à la dernière page de résultats de la recherche courante si cette dernière existe.

asc + <nom du champ de la table de la base de donnée (ou nom de champ attribué quand le champ est un champ additionnel)>

Indique que l'utilisateur désire classer les résultats de manière croissante sur le champ <nom du champ> de la table de la base de données.

desc + <nom du champ>
Indique que l'utilisateur désire classer les résultats de manière décroissante sur le champ <nom du champ> de la table de la base de données.

Variables POST liées aux cases à cocher

chkAll
La case à cocher principale permettant de cocher ou décocher l'ensemble des cases à cocher des résultats visibles dans la zone de résultats.

chkext + <numéro de la ligne courante, commençant à 0>
Les cases à cocher secondaires permettant de sélectionner les lignes de résultat à utiliser pour certaines opérations (par exemple, pour l'export des lignes sélectionnées vers le format d'échange CSV).

Variables POST de champs de type valeur exacte du formulaire

<nom du champ>
Contient la valeur exacte saisie par l'utilisateur pour le critère de recherche courant.
Exemple : num_parcelle pour le champ num_parcelle de la table parcelles.

Variables POST de champs de type liste de valeurs

Idem champs de type valeur exacte.
Contient la valeur choisie par l'utilisateur pour le critère courant dans la liste déroulante.

Variables POST de champs de type classe de valeurs

rd + <nom du champ>
Indique le choix de l'utilisateur entre la possibilité de recherche sur une classe de valeurs majorée ou minorée ou égale à une valeur (par prédicat Egal à, Inférieur à, Supérieur à) et la possibilité de recherche par classe de valeurs majorée et minorée (entre une valeur minimale et une valeur maximale définies).
Dans le cadre d'une recherche de type majorée ou minorée ou égale, cet attribut aura comme valeur « value » alors que dans le cas inverse il aura la valeur « class ».

cmp + <nom du champ>

Si l'utilisateur a opté pour une recherche majorée ou minorée ou égale, indique le prédicat choisi par ce dernier.

<nom du champ>
Si l'utilisateur a opté pour une recherche majorée ou minorée ou égale, indique la valeur minimale ou maximale de cet intervalle borné positivement, négativement ou pas du tout dans le cadre d'une valeur exacte.

inf + <nom du champ>
Si l'utilisateur a opté pour une recherche bornée positivement et négativement, indique la borne inférieure de la classe de valeurs à sélectionner.

sup + <nom du champ>
Si l'utilisateur a opté pour une recherche bornée positivement et négativement, indique la borne supérieure de la classe de valeurs à sélectionner.